Notes - Design - Cognitive Load

- Dr Nick Hayward

A brief intro to cognitive load relative to application and interface design.

Contents

Intro Let us consider, for a moment, the physical act of interacting with a computer, using a mouse or keyboard for example, or touching, swiping, even shaking, a mobile device. Our user's actions may include pressing keys and key combinations, typing repetitive sequences for a prolonged period of time, accurately aiming and moving a given pointing device, and so on. Performing each of these actions naturally incurs a cost of time, and, to varying degrees, both physical and mental effort.

Cognitive load refers to the mental taxation exerted on our user whilst performing a given task. Therefore, it is considered a way of referring to how much sustained attention and cognitive effort is required to perform a task.

In essence, the more complex the task, i.e. the larger the number of contextual details a given user must retain in working memory, the more the task demands a high level of focused attention, and, therefore, the higher the cognitive load becomes for that task.

As you might imagine, it is a good design strategy to try to reduce the user's cognitive load as much as possible. Sounds obvious, but by simply reducing the amount a user has to think about an interface and interaction, the general nuts and bolts of interacting, using, and navigating an application, we reduce the amount of cognitive load we place on our users.

Don't make me think by Steve Krug presents a good introductory overview on this topic.

Impact of interactions Other aspects of interface interaction also require time and effort on the behalf of our user.

For example,

- scrolling, navigating, searching within an application
- choosing options such as menus, lists, forms…
- reading instructions, labels, titles…
- switching contexts (e.g. switching between windows, tabs, pages…)
- switching visual attention (e.g. reading some text, then referring to an image, then looking back at the text, and so on..)
- memory recall for a specific ID, name, action, task sequence…
- simply waiting for the system or application to respond…
- recovering from a specific distraction, such as an interruption not relevant to the current task at hand…

As designers, we need to be aware and mindful not to waste our user's time with unnecessary distractions and requirements within our application. Obviously, we need to try and eliminate unnecessary user actions, thinking, and waiting within our application.

Thinking So, to reduce these cognitive loads we also need to be aware of the types of thinking a user may need in order to interact and engage with our applications.

For example, users may need to think and consider about the following aspects,

- working out the next step in a procedure
- using working memory to help complete an ongoing task
- recall of commands, facts, procedures from long-term memory
- memorising commands, facts, procedures &c. for long-term memory
- referencing information from another source
- making decisions or considering judgements
- mental integration of information from disparate sources, including research, reference, or simply general peripheral sources…

So, how might we immediately start to consider reducing cognitive load for our users?

- use default, standard actions for accessing menus, search, movements such as OS specific swipes, zoom &c.
    - lessens learning of new patterns &c.
- incrementally increase information for an application
    - allow a user to dictate the content, data flow for their experience within an app…
- start to customise the experience as the user uses the app more and more…
    - adjust relative weighting of importance for content, options, menus…

and so on.

Forced, unnecessary thinking Naturally, we do not want to discourage thinking relative to our application, but intellectual thinking and processing is different from forced, unnecessary thinking due to poor design decisions or implementation.

In fact, our application and interface should promote and facilitate the very activity or act of thinking, and record the results where applicable.

For example, we might be encouraging or enhancing the following:

- active research activities with the intention of creating, discovering, or synthesising new knowledge
- creative development and output, including the generation of ideas and content, such as music, video, writing, and so on
- general problem solving and issue resolution
- reading, note taking, and other general tasks…

Our application and interface should not be reducing the cognitive load of these activities themselves, but we can aim to reduce the load exerted by a user's interaction and usage of the interface whilst performing such activities. This is where focus on reliable software, clear and concise output, and task support becomes the driving force for our interface and application.

Cognitive load will, therefore, be reduced by an application's focus upon the task in hand, relevancy of UI information and implementation, and a reduction in extraneous content. So, whilst we can't write the next great novel for our user, we can certainly remove some of the complexity, confusion, and clutter traditionally associated with such application interfaces.

Reduce the amount of additional thinking to the primary task, including better contextual support and research, and we'll provide an application that is easier and more enjoyable for our users.

Quantify cognitive load As designers, developers, and builders we are, of course, interested in how we can quantify the cognitive load involved and required for performing a given task. A better understanding of some of the load issues within our application and interface helps guide us in apportioning emphasis and control throughout our design.

So, for a particular task, we could compile a list of the required actions, operations, or steps that one of our users might require for that task under normal circumstances. We might then estimate or assign a percentage, or some other arbitrary value, which represents our understanding of the effort involved for an individual action. Then, we'd total all of the action scores to assign an overall score for the effort for the selected task. We could then evaluate different design options and alternatives by comparing these collated scores.

The *Keystroke-Level Model for the Goals, Operators, Methods, and Selection Rules* analysis approach, or KLM-GOMS model, (*Card et al, 1983*), is one example of an analysis technique based on this concept. However, instead of assigning scores representing effort, an estimate of the time required for each action is assigned instead. The time required to complete a task is considered a good proxy for physical effort. However, one notable issue is that it doesn't accurately measure the intensity of expended mental effort.

KLM-GOMS model So, a user will accomplish a goal by dividing it into a series of tasks. For each task grouping, our user will take a moment to build their mental representation and then choose an appropriate strategy for accomplishing the task at hand. This preparation is called the *task acquisition time.* Naturally, it can vary from very short periods for simple, routine tasks, to much longer periods, perhaps a few minutes, for more creative, original or thought provoking tasks.

Following this *task acquisition*, our user will then continue with their chosen task using a sequence of actions or operations. The total required time to complete the actions is called the *task execution time.* Therefore, the total time required by our user to complete a chosen task will be the sum of the initial *task acquisition time* and the *task execution time.*

n.b. There are more recent examples of modified models for mobile devices, such as phones, but they are often derived examples as parallels to the original Keystroke-level model.

usage To reach an approximate estimation for the task execution time, KLM-GOMS defines basic operations as follows.

n.b. we'll stick with a simple assumption of our system using a keyboard and mouse option.

Code	Operation	Time (in seconds)
K	Key press & release (keyboard)	Best Typist (135wpm) = 0.08
		Good Typist (90 wpm) = 0.12
		Avg. Skilled Typist (55 wpm) = 0.20
		Poor Typist (40 wpm) = 0.28
		Typing Random Letters = 0.50
		Typing Complex Codes = 0.75
		Worst Typist = 1.20
P	Point mouse to an object on screen	1.10
B	Button press or release (mouse)	0.10
H	Hand from keyboard to mouse & vice-versa	0.40
M	Mental preparation (operation)	1.20
T(n)	Type string of characters	n x K seconds

wpm = words per minute

Source: Kieras, D. 1993. Wikipedia

example So, let us review an example implementation for the KLM-GOMS model.

If we consider an example of finding all of the Latin *et* words in a text document, we may conceptually open the document within a basic text editor.

- move mouse to *search* menu (requires hand to mouse, mental preparation, and point mouse to an object on the screen)
- select the *search* menu from the editor's main menu (requires button press and release of the mouse)
- click on the *find text* link in the *search* menu (requires mental preparation and point mouse to an object on the screen, button press and release of the mouse, and hand from mouse to keyboard)
- enter the required search term, for example *et* (requires key press and release for two characters…)
- click the *OK* or *search* button (requires hand from keyboard to mouse, mental preparation and point mouse to an object on the screen, and a click and release of the mouse)

This sequence can be encoded and expressed using the KLM-GOMS model, thereby allowing us to estimate the required average time for such a task.

Our result might resemble the following table.

Example implementation - text search including mental operators

Action	KLM-GOMS Code	Time (in seconds)
move mouse to **search** menu	H (hand to mouse)	0.40
	M + P (search menu)	1.20 + 1.10
select **search** menu…	BB (select search menu)	2 * 0.10
click on ***find text*** link…	M + P (find text menu item)	1.20 + 1.10
	BB (select menu item)	2 * 0.10
	H (hand from mouse to keyboard)	0.40
enter search term ***et***	KK (type ***et*** characters)	2 * 0.20 (avg. typist)
click the ***OK*** button	H (hand from keyboard to mouse)	0.40
	M + P (***OK*** button)	1.20 + 1.10
	BB (click button)	2 * 0.10
Total		9.10

BB = double button press to simulate mouse click and release (0.20 seconds)

You can see that the results are noticeably influenced by the type of user involved, and we would expect a more skilled or experienced user to complete such tasks in less time. This might be achieved by greater familiarity with the application or perhaps by employing a series of keyboard shortcuts. What is interesting is the comparison of these different paths and options to determine which is quickest, and which is best for the majority of our users. In effect, it becomes our default path.

There are also obvious limitations to such methodologies. It provides only a general rough estimate, and it naturally assumes that users know the correct sequences of actions in order to complete a given task. Also, it does not take into account user errors, application issues, processing considerations, and so on.

However, as designers, models such as KLM-GOMS give us a repeatable way to compare the efficiency of different interface and application alternatives. All other things being equal, the alternative that can be done in the least amount of time will tend to be the most convenient for our users, and thereby create the lowest cognitive load.

Reducing Cognitive Load So, we've now reached the point where we can consider a few tips and tricks in our ongoing efforts to reduce user cognitive load.

For example,

- consistent use of icons, labels, names, and general visual presentation

    - consistency should include design for multiple tasks as well
- clear navigation for process steps…e.g. wizards, paged results… (again, this needs to be consistent across an application, site &c.)
- include visual cues and clues, where possible, to save the user having to remember functionality within an app.
    - for example, further information links, tooltips &c. consider GitHub's use of icons…
- avoid popups except for explicit intervention reasons, such as warnings, errors &c.
- avoid redundancy in content and rendering (this reduces the need for the user to read, view, and process the same content more than once…)
- relational material should be close to one another…avoid making the user switch tabs, windows &c. to find the data
- identify and remove unnecessary steps (also allow users, experienced or otherwise, to turn off instructions, warnings &c.)
- automate processes, steps &c. where possible (again, offer the user the option to modify or turn off this option)
- reduce delays and latency as much as possible. Where processing is required, and a delay may exist, include progress updates and status warnings to guide and inform the user
- productivity apps should offer initial users the option to view templates, tutorials &c. instead of simply opening a blank document. This can obviously be turned off for advanced users, but it helps less experienced users get started more quickly.
- video and audio tutorials are often a lot easier to follow and understand than text only alternatives…particularly useful for beginners and initial users…
- repetitive data entry by the user can be avoided and an app should not force a user to continually remember such details. An exception might be an explicit security feature where the user has opted-in to a pin or password for certain actions…

flow We can also aim to reduce cognitive load by trying to promote user concentration and what is referred to as *flow*.

The term *flow*, as described and popularised by the psychologist *Mihaly Csikszentmihalyi*, refers to a mental state of being completely focused on an activity.

For example, a user who is in a state of flow may exhibit the following characteristics:

- a user's creativity and productivity are high, and the performance of the activity occurs naturally and unconsciously
- a user experiences deep concentration and immersion in their current activity. In effect, a user is both alert and relatively relaxed
- *Living in the moment* or the sensation of being so engrossed in an activity that a user is simply unaware of the passage of time
- balancing interest and challenge - for example, there needs to be sufficient difficulty in the activity to match a user's skills, and effectively maintain their interest, but not too much to create a sense of dread and inevitable failure. It is also important that the task is not too mundane, otherwise inevitable boredom will set in
- a user is confident and exhibits a sense of control over their current situation
- a user is working progressively towards achieving a specific goal. For example, in games this might be as simple as getting to the next level

TED 2004 - Flow, the secret to happiness

flow states and software So, how do we consider and use *flow* states relative to software design.

- it's unusual for beginners to be able to gain *flow* with an application. It normally requires some level of comfort and competence with the operation of an application before *flow* becomes apparent.
- psychologists tell us that getting into a state of *flow* is often quite difficult, and focused concentration may not always be enough. In apps, reducing general cognitive load can aid in this process.

- as with interruptions in the real world, such as phone calls, email, and sudden noise, extraneous messages, clutter, and visual noise within our interfaces can increase the time required to gain *flow*. Such interface distractions can also break a user's existing *flow*.

interface suggestions for flow We may consider some interface suggestions for *flow*

- reduce interruptions in the interface unless intentional for warnings, errors…
  - non-important modal popups, notifications should be avoided
- keep visual presentation simple
  - bright, loud colours and images are jarring to the user's eye
  - unnecessary, prolonged or repetitive animations are distracting
- sequential navigation should be obvious
  - do not require the user to search the interface for *next*, for example…
- avoid switching between tabs, windows, pages for related information
- saving a document, work &c. should be easy and intuitive for a user
- output and display progress reports for ongoing activities
  - progress bars, spinning wheels, timers…
- offer feedback in a prompt and consistent manner within the interface
- multi-tasking for users is difficult
  - don't ask your users to perform too many interface tasks at once…

Resources

- Card, S.K., Moran, T.P. and Newell, A. *The psychology of human-computer interaction.* Lawrence Erlbaum Associates. 1983.
- Holleis, P. et al. *Keystroke-level model for advanced mobile phone interaction.* CHI' 07. New York, USA. 2007.
- Kieras, D. *Using the Keystroke-Level Model to Estimate Execution Times.* 1993. http://courses.wccnet.edu/~jwithrow/docs/klm.pdf
- Krug, S. *Don't make me think, revisited: A common sense approach to web usability.* 3rd Edition. New Riders. 2014.
- Norman, D. *The Design of Everyday Things.* Basic Books. 2013.